# PandaPy: A Wrapper Around Structured Arrays to Mimic 'structs' in the C Language

**Derek Snow**[1],✉

[1]Alan Turing Institute, British Library, 96 Euston Rd, London NW1 2DB, United Kingdom

**Similar to the original Pandas project, PandaPy is developed to improve the usability of python for finance. Structured data types are designed to be able to mimic 'structs' in the C language, and they share a similar memory layout. The biggest benefit of this approach is that NumPy directly maps onto a C structure definition, so the buffer containing the array content can be accessed directly within an appropriately written C program. This makes PandaPy a strong contender for high frequency trading on small-to-medium datasets. PandaPy currently houses more than 30 functions.**

**Data Structure | Finance | HFT | Pandas | Numpy | Python | Software**
**Correspondence:** ***dsnow@turing.ac.uk***

## Introduction

Structured NumPy are developed to interface with C code to allow for low-level manipulation of structured buffers, such as for interpreting binary blobs. It achieves this by sporting specialized features such as subarrays, nested datatypes, and unions, while allowing for control over the memory layout of the structure. PandaPy takes advantage of the low overhead of structured arrays to enable low latency execution of columnar data. It maintains the full speed of NumPy and the usability of Pandas. When using mixed data types (int, float, datatime, str), PandaPy generally consumes (roughly a 1/3rd) less memory than Pandas and for rows less than 50K have execution speeds get up to 10x to 50x faster to that of Pandas.

## Functionality

A Structured Array is an array of structures. NumPy arrays can only contain one data type, but structured arrays in a sense create an array of homogeneous structures. This is done without moving out of NumPy, such as is required with Xarray. For structured arrays the data type only has to be homogeneous within each column like an SQL database. Additionally, each column can be another multidimensional object and does not have to conform to the basic NumPy datatypes. It can include nested datatypes, subarrays, functions as long as each column conforms to the array length, this allows for a great amount of flexibility.

PandaPy wraps these structures in vectorised functions to allow for similar functionality as seen in packages like Pandas, such as groupby, pivot, sort, and others. If you find yourself writing a Python interface to a legacy C or Fortran library that manipulates structured data, you'll probably find structured arrays quite useful.

Machines are getting large, EC2 X1 has 2TB of RAM and is remarkably affordable. If your execution can be done on a single machine then it should be done on a single machine. Even developers working on large-scale distributed data processing software like Dask claims that it is often faster and easier to use Pandas (PandaPy, NumPy) within memory than to use software like Dask or Apache Spark.

## Performance

Maintains the full functionality and speed of structured NumPy (eg., array[col1] + array[col2], same speed as numpy vector_1 + vector_2). Pandas outperforms PandaPy at the same point when Pandas outperforms NumPy. NumPy generally performs better than pandas for 50K rows or less. Pandas generally performs better than numpy for 500K rows or more; from 50K to 500K rows it is a toss up depending on the operation. Because both Pandas and PandaPy is built on NumPy, the performance difference can be attributed to Pandas overhead. For larger datasets Pandas' hash tables and columnar data format gives it the upperhand on many operations.

There are however many PandaPy operations that improve relative to Pandas as the number of rows increase: rename, column drop, fillna mean, correlation matrix, value filter (array > 0), value reads (a=array[col]), singular value access (array[col][pos]), atomic functions (sqrt, power). If you need Pandas for speciality functions, you can also simply move back and forth with: df = pp.pandas(array) and back array = pp.structured(df).

For simple calculations on a small dataset (i.e, plus, mult, log) PandaPy is 25x - 80x faster than Pandas. For table functions (i.e., group, pivot, drop, concat, fillna) on a small data set PandaPy is 5x - 100x times faster than Pandas. For most use cases with small data, PandaPy is faster than Dask, Modin Ray, and Pandas. You can have a look at the interactive notebook to get a better insight into the speed differences [1]. PandaPy is not created to allow you to scale up to clusters for multiple computer processing like Dask, Modin, and Spark, instead it is focused on speed and usability within a single computer's Memory.

## Package

In this section the functions are organised according to the following subtitles: structure, explorative, finance, and array categories. The function description is given e.g., "Read in Arrays", and then the function name follows, e.g., "read",

[1]https://colab.research.google.com/drive/1i9vZvOQDgvjRyh_dME5WV143s0ndPSjU?usp=sharing



Electronic copy available at: https://ssrn.com/abstract=3599639

and where appropriate, the speed difference between PandaPy and Pandas is given as achieved on stock market data tested on an interactive Notebook e.g., "3X".

**Array Structure.** The first few functions are created to read data directly from urls and allows you to move between Pandas and PandaPy as well as between structured (homogeneous) and unstructured (diverse) arrays.

```
Read In Arrays (read)
To Pandas (unstructured)
Pandas to Structured (structured)
To Unstructured (to_unstruct)
To Structured (to_struct)
Print Table (table)
```

**Explorative Functions.** The explorative techniques aim to replicate and expand on the describe function available in Pandas as well as the correlation matrix.

```
Descriptive Statistics (describe) (5x)
Correlation Array (corr) (2x)
```

**Finance Functions.** To showcase a few financial examples, I have created new function that take in market data for a portfolio to calculate returns, lagged returns and portfolio value.

```
Returns (returns) (50x)
Column Lags (lags) (7x)
Portfolio Value (portfolio_value) (50x)
```

**Array Functions.** Here we add functionality to everything you would ordinarily expect from accessible data processing software. Generally PandaPy does much better in this domain than Pandas.

```
Drop Null Rows (dropnarow) (30x)
Drop Column/s (drop) (100x)
Add Column/s (add) (3x)
Concatenate (concat) (70x)
Merge (merge) (2x)
Group by (group) (10x)
Pivot (pivot) (20x)
Fill Nulls (fillna) (20x)
Shift Column (shift) (50x)
Rename (rename) (500x)
```

**Other Tests.** Here we just test a few more operations like assigning a new value, adding, multiplying, and taking the log to finalise the speed tests.

```
Update (a[col] = values) (60x)
Add (a[col] + a[col]) (80x)
Multiply (a[col] * a[col]) (80x)
Log (np.log(array[col]) (25x)
```

## Conclusion

If you head to the GitHub[2] repository you would see links to interactive notebooks that would allow you to experiment the speed difference on your own unique dataset. Current feedback suggests that the tool is popular in low latency environments. One user in the retail sector reports that PandaPy is the best solution they have seen to deal with their online latency issues. They run a dynamic pricing algorithm which ingests around 380 seconds per median session with around 270 feature values.

[2] https://github.com/firmai/pandapy


Electronic copy available at: https://ssrn.com/abstract=3599639